Can I build an Embedded Linux system with Clang

Khem Raj

#lfelc @himvis

Agenda

- Introduction
- Clang based Toolchain
- Kernel Status
- Platform Build using Yocto Project
- Select Compiler Runtime (C/C++)
- User Space
- Common Errors

Building Platform

- LLVM Runtime
  - compiler-rt
    - Compiler built-ins
    - Sanitizer runtimes
    - Profile
  - libc++
    - libc++ - C++ Standard Library Support
    - libc++abi - low level support for a standard C++ library
  - libunwind
    - LLVM's unwinder library

Tools Map

Component	LLVM	GNU
C/C++ Compiler	clang/clang++	gcc/g++
Assembler	cc1as	as
Linker	lld	ld, gold
Debugger	lldb, lldb-server	gdb, gdb-server
Compiler runtime	compiler-rt	libgcc
Unwinder	libunwind	libgcc
C++ runtime	libc++, libc++abi	libstdc++, libsupc++
Std C library	llvm-libc	glibc
Binutils	llvm-ar, llvm-nm, llvm-ranlib …	Ar, nm, ranlib …

Additional Clang Tools

- clang-tidy – C/C++ linter tool
- clang-doc – Generate documentation
- clangd – For adding features to editors
- scan-build – Static Analyzer

Tools

- Clang advertises itself as GCC 4.2.1

```
% clang -dD -E -x c /dev/null | grep GNUC
#define __GNUC__ 4
#define __GNUC_MINOR__ 2
#define __GNUC_PATCHLEVEL__ 1
#define __GNUC_STDC_INLINE__ 1
```

```
% clang -dD -E -x c /dev/null | grep clang
#define __clang__ 1
#define __clang_major__ 10
#define __clang_minor__ 0
#define __clang_patchlevel__ 1
#define __clang_version__ "10.0.1 "
```

- Clang Assembler
  - Supports Unified syntax only
  - No symbol calculations
  - Disable with `-fno-integrated-as`

Kernel

- Upstream Kernel is buildable with clang
  - Landing Page
    - https://clangbuiltlinux.github.io/
  - CI status – Uses Travis
    - https://travis-ci.com/github/ClangBuiltLinux/continuous-integration
  - Issue Tracker – Github Issues
    - https://github.com/ClangBuiltLinux/linux/issues

ClangBuiltLinux

Building the Linux kernel with Clang

View My GitHub Profile

Useful links

build passing

- Official Kernel Docs
- Issue tracker
- Wiki
- Repos
- Mailing List: `clang-built-linux@googlegroups.com` (archive)
- IRC: `#clangbuiltlinux` on chat.freenode.net (webchat)
- Telegram: @ClangBuiltLinux
- Bi-weekly video meeting
  - Calendar
  - Hangouts Meet

The following architectures are targetable from both LLVM and the Linux kernel and are relatively well supported and tested:

- arm
- arm64
- x86

THE LINUX FOUNDATION

Kernel

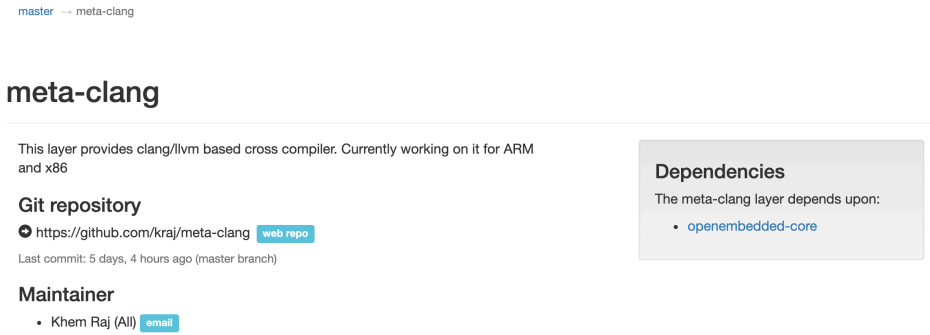
- Well Supported
  - ARM
  - AARCH64
  - X86
- Limited Test Configurations
  - Powerpc
  - Mips
- In progress
  - RISC-V
- Add yours …

Building Platform

- There are few options
  - Gentoo has clang overlay
  - Debian
  - Mageia
  - Yocto Project/OpenEmbedded
  - DIY…
- Here Yocto Project approach is used

Using Clang in Yocto Project

- Yocto Project Layer
  - meta-clang
  - Provides recipes for clang cross compiler and tools e.g. lldb,lld
- https://github.com/kraj/meta-clang



Building Platform

- Yocto Project Setup with Clang

```
$ git clone git://git.yoctoproject.org/poky
$ cd poky
$ git clone git://github.com/kraj/meta-clang
$ . ./oe-init-build-env
$ bitbake-layers add-layer ../meta-clang
```

Building Platform

- Set Clang as default compiler

```
TOOLCHAIN = "clang"
```

- Set compiler runtime to use LLVM runtime
  - Uses compiler-rt, libc++, llvm libunwind

```
RUNTIME = "llvm"
```

Building Platform

- Build image

```
$ bitbake core-image-sato
```

- Run image

```
$ runqemu
```

- Build SDK

```
$ bitbake core-image-sato –cpopulate_sdk_ext
```

Building Platform

- Exceptions to building with clang
  - https://github.com/kraj/meta-clang/blob/master/conf/nonclangable.conf
  - Some are just flag tweaks
  - Some override compiler to always be GNU Compiler
    - TOOLCHAIN = “gcc”

Building Platform

- Exceptions to building with clang
  - GLIBC
    - Depends on GCC features
      - Musl C library works fine
  - GCC runtime – Needs GCC to compile itself
  - U-boot – Some configs do work
    - https://github.com/u-boot/u-boot/blob/master/doc/build/clang.rst
  - Elfutils – Contains GNU'ism
  - Grub – Experimental support
    - Git version compiles with CFLAGS="-Wno-error"
  - Python3 - Qemu can't run profile tests run during build
  - Many packages do not build with Clang assembler
    - Uses `-no-integrated-as`
  - In some cases inline asm is not understood by clang

Building Platform

- C Runtime (crt)
  - Providers include libgcc and compiler-rt
  - Yocto default uses these objects from libgcc
    - crtbegin.o/crtend.o
  - Enable by adding ‘crt’ to PACKAGECONFIG

```
PACKAGECONFIG ??= ""
PACKAGECONFIG[crt] = "-DCOMPILER_RT_BUILD_CRT:BOOL=ON,-DCOMPILER_RT_BUILD_CRT:BOOL=OFF"
```

Building Platform

- Choosing Runtimes
  - Using GNU runtime works well
  - Mixing both may not
    - Yocto's package specific staging helps
  - Using libc++ at system level
    - Does not work for recipes pinned to use gcc

Building Platform

- Using LLVM LLD Linker
  - LLD is built but not turned on as system linker
  - Use via `--fuse-ld=lld`
  - Default can be set via `ld-is-lld` in DISTRO_FEATURES
- AR, RANLIB, NM
  - Uses llvm versions when TOOLCHAIN = “clang"

Platform Build

- Using LTO
- `inherit lto`
- Exposes `thin-lto` and `full-lto` via DISTRO_FEATURES

Platform Build

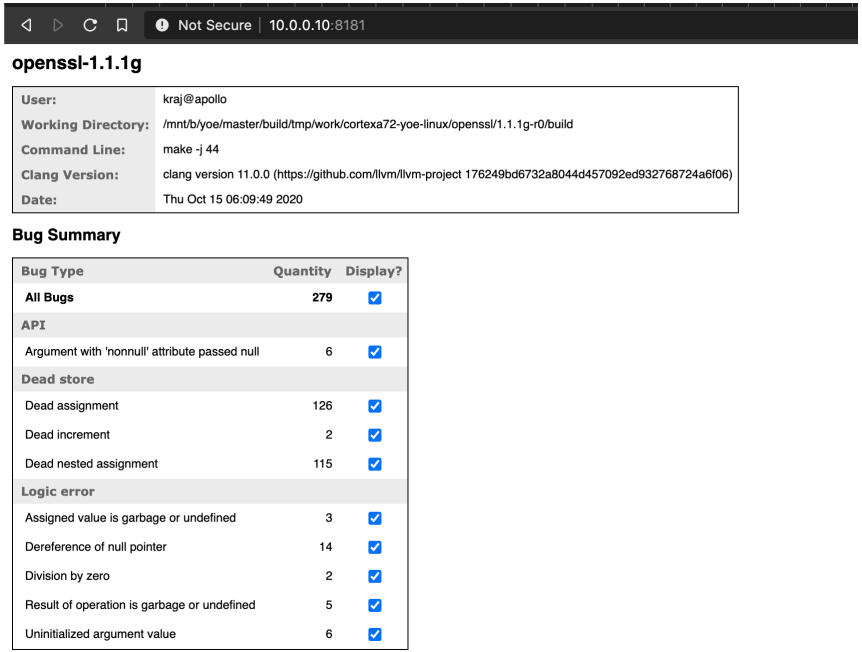
- Static Analyzer
  - Enable in local.conf

```
INHERIT += "scan-build"
SCAN_BUILD ?= ""
SCAN_BUILD_pn-curl = "1"
```

  - Disable for given recipe

```
SCAN_BUILD_pn-<recipe> = “”
```

  - View results
  - `bitbake –c scanview <recipe>`



Platform Build

- Installing Extensible SDK

```
/mnt/b/yoe/master/build/tmp/deploy/sdk/yoe-x86_64-yoe-sdk-image-cortexa72-raspberrypi4-64-toolchain-ext-3.2.0-beta.sh -y -d /mnt/b/yoe/yoe_sdk/3.2.0-beta
```

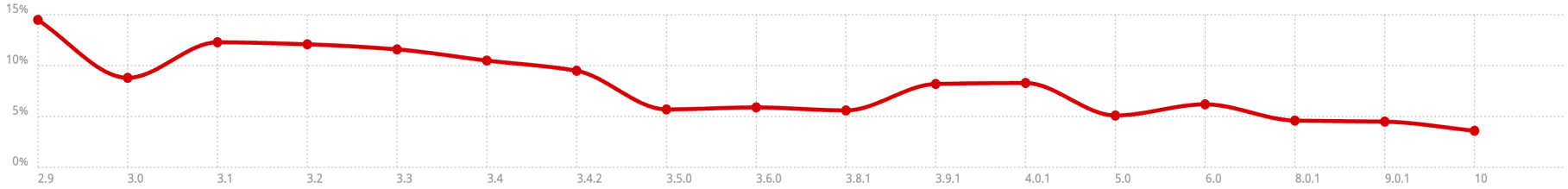
- Using SDK

```
% . /mnt/b/yoe/yoe_sdk/3.2.0-beta/environment-setup-cortexa72-yoe-linux
SDK environment now set up; additionally you may now run devtool to perform development tasks.
Run devtool --help for further details.
```

- Clang specific Env variables
  - CLANGCC, CLANGCXX, CLANGCPP, CLANG_TIDY_EXE

Platform Build

- Debian
  - https://clang.debian.net/
  - Rebuils Results with Clang 10
    - 31014 packages tried. Among them, 1110 (3.6 %) failed.



Common Errors

- imake failure
  - Expects traditional GCC specific pre-processor behavior (-traditional-cpp)

Common Errors

- C++11 requires a space between literal and identifier
  - Can be suppressed disabling -Wreserved-user-defined-literal

Common Errors

- Link with LTO fails
  - CC passed to gold plugin should have absolute paths

```
/usr/bin/ld: /usr/lib/llvm-
10/bin/../lib/LLVMgold.so: error loading
plugin: /usr/lib/llvm-
10/bin/../lib/LLVMgold.so: cannot open shared
object file: No such file or directory clang:
error: linker command failed with exit code 1
(use -v to see invocation)
```

Common Errors

- Missing symbols at link time
  - Clang follows C99 ‘inline’ behavior by default
    - https://clang.llvm.org/compatibility.html#inline
  - Older versions of GCC defaulted to `-gnu89`

Common Errors

- Security: Format string is not a string literal
  - Clang errors **printf** style format here
  - Fails to compile with clang but not with gcc

```
#include <stdio.h>
void foo(void) {
  char buffer[1024];
  sprintf(buffer, "%n", 2);
}
```

```
error: format specifies type 'int *' but the argument has type 'int' [-Werror,-Wformat]
   sprintf(buffer, "%n", 2);
                    ~~   ^
1 error generated.
```

Summary

- Clang can be used as default system compiler
  - GCC is still needed for glibc, GNU runtime
  - U-boot – Some configs can be compiled
  - Musl is ok

Thank you for your time

THE LINUX FOUNDATION

Embedded Linux Conference Europe